Extensión del Modelo Linda para representar la Dinámica de Conocimiento en Sistemas Multi-agente*

Marcelo A. Falappa Alejandro J. García

Instituto de Investigación en Ciencia y Tecnología Informática (IICyTI)
Departamento de Ciencias e Ingeniería de la Computación (DCIC)
Universidad Nacional del Sur (UNS)
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Av. Alem 1253 - (B8000CPB) Bahía Blanca - Argentina
Phone/Fax: (+54)(291)459-5136
E-mail: [mfalappa,ajg]@cs.uns.edu.ar

Palabras Clave: Dinámica de Conocimiento, Inteligencia Artificial, Sistemas Multi-agente, Sistemas Distribuidos.

Resumen

Este trabajo es un primer acercamiento al estudio de las relaciones existentes entre el modelo Linda y la Teoría de Cambio de Creencias. Se mostrarán las similitudes existentes entre ambos formalismos y cómo extender el modelo Linda utilizando resultados y técnicas del área de Teoría de Cambio de Creencias.

En tal sentido, se establecen convenciones para representar conocimiento en el espacio de tuplas de Linda, y se formulan extensiones a las primitivas provistas por el lenguaje para modelar la dinámica de conocimiento de los agentes en un sistema multi-agente.

1. Introducción

Los sistemas basados en conocimiento permiten almacenar gran cantidad de información sobre determinadas aplicaciones en un lenguaje particular, y obtener respuestas sobre el estado del sistema en dicho lenguaje, contando para ello con mecanismos de inferencia que permiten derivar conocimiento. Muchos de estos sistemas permiten a su vez modificar el conocimiento para representar la dinámica del mundo en que se desenvuelven. Estos tipos de cambio se asemejan a las actualizaciones en bases de datos aunque,

*Financiado parcialmente por CONICET (PIP 5050), Universidad Nacional del Sur (24/ZN09), y Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro 13096.)

dado que la misma contiene conocimiento, se deben recurrir a avanzadas técnicas de actualización desarrolladas en el área de *dinámica de conocimiento*.

Por otra parte, los sistemas multi-agente son sistemas en los cuales muchos agentes pueden interactuar entre sí. Usualmente estos agentes cuentan con una base de conocimiento la cual utilizan para razonar acerca del mundo. Esta base de conocimiento es dinámica y debe ser actualizada como consecuencia de percepciones del agente o interacciones con otros agentes del sistema. Por otra parte, el conocimiento de los agentes puede estar fragmentado o replicado (distribuido) en la red de agentes.

El objeto de este trabajo es establecer relaciones entre ciertas técnicas de programación paralela mediante el uso del paradigma Linda para la sincronización de agentes en un sistema multi-agente. El artículo está organizado como sigue. En la Sección 2 se presenta el modelo Linda y una posible implementación. En la Sección 3 se presentan las tres operaciones básicas de la teoría de cambio de creencias: expansiones, contracciones y revisiones, con sus propiedades características y sus posibles aplicaciones. En la Sección 4 se presentan las relaciones entre operaciones primitivas de C-Linda y dos de las operaciones básicas de la teoría de cambio de creencias: expansiones y contracciones. En la Sección 5 se presenta una extensión a C-Linda que permite modelar operaciones de contracción múltiple así como operaciones de revisión. Por último, en la Sección 6 se presentan posibles aplicaciones, y en la Sección 7 las conclusiones y el trabajo futuro.

2. El modelo Linda

2.1. Operaciones Básicas

Linda [CG92b, CG92a] es un modelo para programación paralela que permite la comunicación y sincronización de procesos, mediante el agregado o la eliminación de elementos en un área de memoria compartida denominada *espacio de tuplas*.

Un lenguaje base con el agregado de las operaciones sobre el espacio de tuplas provisto por Linda, deriva en un dialecto de programación paralela. Linda es un modelo de creación y coordinación de procesos que es *ortogonal* al lenguaje base en el que está embebido. El modelo Linda no considera *como* se computan los múltiples threads de ejecución; simplemente considera como esos threads de ejecución son creados y organizados de una manera coherente.

El modelo Linda es un modelo de memoria que utiliza un espacio de tuplas de dos tipos: *tuplas de procesos* y *tuplas de datos*. Las tuplas de procesos (que se ejecutan todas simultáneamente) intercambian datos generando, leyendo y consumiendo tuplas de datos. Una tupla de procesos que está finalizando su ejecución se transforma en una tupla de datos que es indistinguible del resto.

Linda soporta cuatro operaciones básicas sobre el espacio de tuplas: `out`, `in`, `rd` y `eval`:

- `out`(t) agrega la tupla t al espacio de tuplas.
- `eval`(t) es similar a `out`(t) salvo que agrega la *evaluación* de t al espacio de tuplas. La evaluación de t crea un proceso nuevo para cada campo de t. Cuando todos los

campos han sido evaluados, la tupla t se transforma en una tupla pasiva que puede ser leída y manipuada como cualquier otra.

- `in`(t) permite que una tupla s que concuerde[1] con t sea eliminada del espacio de tuplas, teniendo dos posibilidades:

  a) Si no existe ninguna tupla s que concuerde con t, el proceso se suspende hasta que exista una tupla.

  b) Si existen varias tuplas que concuerden con t, se elimina una tupla al azar.

- `rd`(t) es similar a `in`(t), salvo que no se borra la tupla del espacio de tuplas, permitiendo leer una tupla sin tener que eliminarla.

Las operaciones `in` y `rd` tienen dos variantes: `inp` y `rdp` (`p` de predicado) respectivamente:

- `inp`(t) es similar a `in`(t) salvo que si falla en encontrar una tupla que concuerde con t retorna un valor 0; de lo contrario, retorna un valor 1.

- `rdp`(t) es similar a `rd`(t) salvo que si falla en encontrar una tupla que concuerde con t retorna un valor 0; de lo contrario, retorna un valor 1.

Con las operaciones antes mencionadas, es posible implementar sobre el espacio de tublas primitivas de sincronización como semáforos o estructuras de datos tradicionales como arreglos, matrices, registros, pilas, colas, etc. Por ejemplo [CG92b], para ejecutar una operación *wait* sobre un semáforo `sem` se ejecuta la operación:

```
in(sem)
```

Análogamente, para ejecutar la operación *signal* sobre el semáforo `"sem"` se ejecuta la operación:

```
out(sem)
```

2.2. Implementaciones de Linda

Actualmente existen varias implementaciones de este modelo, las cuales están embebidas en diferentes lenguajes de programación: C, Prolog, Lisp, etc.

Por ejemplo, C-Linda [CG92b] es una implementación del modelo Linda que soporta programación paralela. El espacio de tuplas en C-Linda es una serie de valores tipados como, por ejemplo, `("si",24.75,12,x)` o `("no",? f,? d)`. Las tuplas pueden ser *valores* o *variables*. Los valores pueden ser alfanuméricos o numéricos; las variables son objetos sin especificar y van precedidas del símbolo `?`. Por ejemplo, la matriz:

$$A = \begin{pmatrix} 10 & 20 \\ 30 & 40 \end{pmatrix}$$

puede representarse mediante las siguientes tuplas:

[1] Entendemos por concordancia a un proceso de *pattern matching* o *unificación* entre términos.

```
(A,1,1,10)
(A,1,2,20)
(A,2,1,30)
(A,2,2,40)
```

Luego, por ejemplo, para leer el elemento ubicado en la componente [1,2] se puede hacer `rd(A,1,2,?x)`. En cambio, para asignarle un nuevo elemento con valor 100, a esa misma componente, se debe ejecutar primero `in(A,1,2,?x)` seguido de `out(A,1,2,100)`.

También existen implementaciones del modelo Linda en programación en Lógica como en Shared Prolog [ACD90] o BinProlog/Jinni [Tar, Tar99]. En estas implementaciones una tupla es un término Prolog, y las variables tienen las convenciones denotacionales de Prolog. El mecanismo de unificación es utilizado tanto para almacenar, como para recuperar o eliminar tuplas. Por ejemplo, la componente [1,2] de la matriz A indicada arriba se puede almacenar como el término `matrix(a,1,2,20)`, para recuperar el valor de esa componente se haría `rd(matrix(a,1,2,X))`, y para cambiar su valor `in(matrix(a,1,2,_))` seguido de `rd(matrix(a,1,2,100))`.

3. Teoría de Cambio de Creencias

Los sistemas de revisión de creencias son sistemas lógicos para modelar la dinámica de conocimiento, esto es, cómo modificar nuestras creencias ante el arribo de información nueva. El problema surge cuando esta información es inconsistente con las creencias que representan nuestro estado de conocimiento. Puesto que es deseable preservar consistencia en el estado de conocimiento, generalmente es necesario eliminar ciertas creencias preservando tanta información original como sea posible (principio de *mínimo cambio* [AGM85, Gä88]).

El modelo AGM define tres tipos de operaciones de cambio: expansiones, contracciones y revisiones. Las expansiones se definen mediante operadores de consecuencia y uniones de conjuntos por lo que su definición es directa. En cambio, las contracciones y las revisiones requieren eliminar creencias del conocimiento de un agente por lo que es necesario contar con algún *mecanismo de selección* para determinar que sentencias serán eliminadas. Las contracciones en el modelo AGM se denominan *partial meet contractions* y están basadas en la selección de los subconjuntos maximales que no implican la información a ser eliminada. Para hacer esto, las partial meet contraction usan *funciones de selección*.

Asumiremos un lenguaje al menos proposicional $\mathcal{L}$ con los conectivos $\neg, \rightarrow, \wedge, \vee$, y un operador de consecuencia Cn que satisface *inclusión* ($A \subseteq Cn(A)$), *iteración* ($Cn(A) = Cn(Cn(A))$) y monotonía (si $A \subseteq B$ entonces $Cn(A) \subseteq Cn(B)$). A partir del operador de consecuencia Cn podemos definir una relación de derivación $\vdash$ tal que $\alpha \in Cn(A)$ si y solo si $A \vdash \alpha$.

Sea K un conjunto de sentencias de $\mathcal{L}$ que es consistente. En el modelo AGM pueden adoptarse tres actitudes epistémicas:

- **Aceptación**: α es aceptada en K si $\alpha \in Cn(K)$.

- **Rechazo**: α es rechazada en K if $\neg\alpha \in Cn(K)$.
- **Indeterminación**: α es indeterminada en K si $\alpha \notin Cn(K)$ y $\neg\alpha \notin Cn(K)$.

Las operaciones de cambio tradicionales en el modelo AGM son [AGM85, Gä88]:

- **Expansión**: permite agregar una sentencia α a un conjunto K, y se nota como $K{+}\alpha$. Si α es *indeterminada* en K entonces α es *aceptada* en $K{+}\alpha$ y *rechazada* en $K{+}\neg\alpha$.
- **Contracción**: permite eliminar una sentencia α de un conjunto K, y se nota como $K{-}\alpha$. Si α es *aceptada* en K entonces α es *indeterminada* en $K{-}\alpha$; si α es *rechazada* en K entonces α es *indeterminada* en $K{-}\neg\alpha$.
- **Revisión**: permite cambiar radicalmente la actitud epistémica de una sentencia α con respecto a un conjunto K, y se nota como $K{*}\alpha$. Si α es *aceptada* en K entonces α es *rechazada* en $K{*}\neg\alpha$; si α es *rechazada* en K entonces α es *aceptada* en $K{*}\alpha$.

Cada una de estas operaciones satisface una serie de postulados básicos que caracterizan el comportamiento de cada operación. En particular, la operación de expansión es la más simple de definir ya que se define de dos maneras alternativas [Han99]:

Para Bases de Creencias (Belief Bases): $K{+}\alpha = K \cup \{\alpha\}$.

Para Conjuntos de Creencias (Belief Sets):[2] $\mathbf{K}{+}\alpha = Cn(\mathbf{K} \cup \{\alpha\})$.

A partir de estas definiciones puede observarse que las expansiones permiten el agregado **irrestricto** de creencias. Esto es, si α es aceptada en K entonces es aceptada y rechazada en $K{+}\neg\alpha$. Análogamente, si α es rechazada en K entonces es aceptada y rechazada en $K{+}\alpha$. Por tal motivo, las operaciones de expansión no son muy atractivas de aplicar ya que pueden generar fácilmente inconsistencias. Obviamente, un estado de conocimiento inconsistente no es una situación deseable en ningún sistema de razonamiento basado en conocimiento. En cambio, las revisiones, permiten agregar información tratando de preservar consistencia en el conjunto resultante. Para ello, al igual que en las contracciones, es necesario contar con un mecanismo de selección que permita decidir que creencias pueden ser eliminadas en el proceso de cambio para preservar consistencia.

El modelo AGM tiene muchos puntos controversiales y eso ha dado lugar a la formulación de operadores alternativos. Por caso, las operaciones de cambio no priorizadas son operaciones de cambio en los que la nueva operación no siempre es aceptada o es aceptada parcialmente. Entre los modelos alternativos podemos citar las operaciones de cambio con credibilidad limitada [HFCF01], las operaciones de semirevisión [Han97a], las operaciones de mezcla [Fuh97], las operaciones de revisión por conjuntos de sentencias [FKIS02], las operaciones de screened revision [Mak97, Han97b], etc.

[2]Los conjuntos de creencias son conjuntos clausurados bajo un operador de consecuencia lógica. Esto es, $\mathbf{K}$ es un *conjunto de creencias* si y solo si $\mathbf{K} = Cn(\mathbf{K})$. Los conjuntos de creencias suelen notarse con letras mayúsculas negritas.

Otra propuesta son las operaciones de cambio *múltiples*, en las que se permite agregar o eliminar más de una creencia a la vez. En tal sentido, podemos citar las contracciones múltiples en sus dos variantes: *choice contraction* y *package contraction* [FH94]. Básicamente, si se desea contraer un conjunto K con respecto a otro conjunto A, una *choice contraction* permite eliminar de K al menos una sentencia de A que esté en K; una *package contraction* permite eliminar de K todas las sentencias de A que estén en K.

4. Relación entre Linda y Operaciones de Cambio

Es interesante ver como algunas primitivas provistas por Linda se asemejan a operaciones de cambio. Por ejemplo, la operación `out`(t) que permite agregar una tupla t al espacio de tuplas se asemeja a una operación de *expansión*. Por su parte, la operación `in`(t) que permite eliminar una tupla t del espacio de tuplas tiene relación con la operación de *contracción*. Por otro lado, existen diferencias entre los dos formalismos que es interesante estudiar para poder extender ambos formalismos. Por ejemplo, la operación `in`t queda suspendida si no existe una tupla para eliminar que concuerde con t. Por su parte, no existe una versión directa del operador de revisión en Linda.

Puesto que el espacio de tuplas es genérico y no impone restricciones de forma en las mismas, es tarea del programador encargarse de eso. En particular, en un sistema multi-agente se puede utilizar el espacio de tuplas para almacenar el conocimiento de un agente mediante tuplas de datos. Puesto que Linda permite la ejecución paralela, esto permite la interacción de varios agentes en el *mismo* espacio de tuplas. Por lo tanto, varios agentes pueden realizar modificaciones simultáneas en el espacio de tuplas (conocimiento compartido) mediante las operaciones `in` (expansiones) y `out` (contracciones).

Sin embargo, para que se represente conocimiento en el espacio de tuplas es necesario imponer ciertas restricciones sobre las tuplas que representen conocimiento. En particular, debe ser posible agregar al menos un operador de negación ($\neg$ / `no`) así como también implicación ($\rightarrow$ / `->`). Luego, en el espacio de tuplas puede almacenarse conocimiento mediante convenciones notacionales. Por caso, `(bel,gino,¬fly(tweety))`[3] puede representar que Gino cree que Tweety no vuela; `(bel,_,car(ferrari))`, establece que todos los agentes creen que Ferrari es un auto.

5. Nuevas Primitivas para Linda

Como fue explicado en la sección anterior las operaciones `in` y `out` están relacionadas con expansiones y contracciones en un espacio de tuplas. En este trabajo el objetivo es realizar una extensión al modelo Linda para el caso de almacenar conocimiento en el espacio de tuplas. Nuestra propuesta es agregar una operación `rev` que permita realizar revisiones, una operación `out-all` que permita realizar contracciones múltiples del tipo *package contraction*, y una operación `out-some` que permita realizar contracciones múltiples del tipo *choice contraction*.

[3] `bel` indica el concepto de creencia (en Inglés *belief*).

Supongamos que la forma de las tuplas de conocimiento es de la forma `(bel,I,A)` que determina que la creencia `A` es conocimiento del agente `I`. En caso de conocimiento compartido, no se especifica el agente y la tupla es de la forma `(bel,_,A)`.

La nueva operación `rev(bel,I,A)` permite revisar el espacio de tuplas de conocimiento de modo tal que se eliminan las tuplas `(bel,I,¬A)`, esto es, se eliminan las tuplas que establecen que el agente `I` cree la creencia `¬A`.

Esta composición de operaciones está formalmente expresada en la *identidad de Levi* [Gä88, Lev77]:

$$K*\alpha = (K-\neg\alpha)+\alpha$$

Esto es, para revisar con respecto a α primero se elimina cualquier derivación de $\neg\alpha$ para luego incorporar la creencia α. Para ello, necesitamos definir operaciones de *multiple cambio*:

- `out-all`(t): que elimina todas las tuplas que unifican con t.
- `out-some`(t): que elimina un subconjunto de tuplas que unifican con t.

`out-all`(t) equivale a una *package contraction* con respecto a t. `out-some`(t) equivale a una *choice contraction* con respecto a t y requiere algún criterio de preferencia para determinar cuales tuplas que unifican o concuerdan con t son eliminadas. En sus casos límite, `out-some`(t) equivale a `out`(t) si se elimina solamente una tupla; por el contrario, `out-some`(t) equivale a `out-all`(t) cuando se eliminan todas las tuplas que concuerdan con t.

Por ejemplo, `rev`(t) podría definirse como una secuencia de operaciones de la forma:

$$\texttt{out-all}(\neg t);\ \texttt{in}(t)$$

En caso de que t contenga variables, se disparán procesos paralelos que intentarán eliminar todas las posibles instancias de $\neg t$. Cuando todos los procesos terminan, se agrega t derivando en un conjunto consistente. Por ejemplo, `rev(bel,I,A)` permite que el agente `I` crea en `A`, posiblemente eliminando (en paralelo) las posibles derivaciones de `¬A`. Análogamente, `rev(bel,_,A)` permite que **todos** los agentes pasen a creer en `A`, eliminando (en paralelo) las respectivas derivaciones `¬A`.

6. Ejemplo y Aplicaciones

Dado que el espacio de tuplas que utiliza Linda es genérico, es necesario establecer convenciones para representar conocimiento. Para ello, imaginemos que en el espacio de tuplas se representa el conocimiento de 4 agentes que juegan al fútbol como parte del torneo internacional Robocup [Rob].

RoboCup es un proyecto internacional conjunto que promueve el desarrollo de técnicas de inteligencia artificial y robótica cognitiva, con el objetivo de lograr que un equipo de agentes autónomos jueguen al fútbol con determinadas reglas, dependiendo de las categorías.

Uno de los principales desafíos actuales de este dominio de aplicación, es desarrollar agentes autónomos independientes que cooperen entre sí para ganar un partido o una serie de partidos contra equipos equivalentes. Para esto, es importante que los agentes tengan conocimiento sobre las posiciones de los compañeros de equipo, la ubicación de los rivales, la ubicación de la pelota, la dirección de la misma, etc. También, es necesario almacenar conocimiento que determine la forma en que el equipo afrontará el partido: si será agresivo en ataque, si adoptará una táctica defensiva, si realizará marca personal, etc.

A continuación se mostrará un ejemplo de aplicación de nuestra propuesta a este dominio. Para simplificar la presentación del ejemplo, se asumirá que en el espacio de tuplas se almacena conocimiento común (para todos los agentes), y conocimiento específico (de cada uno de los agentes):

```
(bel,_,numeroderivales=4)
(bel,pato,atajar)
(bel,chicho,defender)
(bel,nino,atacar)
(bel,pipo,marcar_1_to_1)
```

Este simple espacio de tuplas, representa conocimiento de los agentes de un equipo determinado, asumiendo que todos los agentes saben que: el equipo rival tiene 4 integrantes, Pato es el agente que debe atacar, el agente Chicho tiene que como estrategia defender, el agente Nino tiene como estrategia atacar, y Pipo es un agente encargado de la marca individual de rivales. Es necesario remarcar el hecho de que existe cierto conocimiento común implícito que debe almacenarse explícitamente: si un agente ataca entonces no realiza ninguna de las demás acciones posibles, y así con el resto de las estrategias:

```
(bel,_,atacar -> no defender)
(bel,_,atacar -> no atajar)
(bel,_,atacar -> no marcar_1_to_1)
(bel,_,defender -> no atacar)
...
```

Supongamos que a los 30 segundos del partido, le marcan un gol al equipo en cuestión. En esa situación, dado que aún resta mucho tiempo por jugar, la estrategia del equipo puede permanecer inalterable, y se decide asignar una estrategia diferente a Nino para que no esté en posición tan ofensiva ya que por su sector se inició la jugada del gol del equipo rival. En ese caso, se podría realizar una operación `rev(bel,nino,marcar)`. Como resultado de esta operación, se eliminará `(bel,nino,atacar)` del espacio de tuplas y agregará `(bel,nino,marcar)`.

En caso de que el equipo tenga una estrategia conservadora, esto es, ningún jugador está decidido a atacar, se puede realizar una operación del tipo:

```
rev(bel,_,atacar)
```

Esta operación modifica la estrategia de todos los agentes, haciendo que cada uno tenga la orden de atacar e, implícitamente, se deben eliminar (en paralelo) las órdenes de

defender, marcar, o atajar. Esta medida parece un poco drástica ya que obligaría incluso al arquero a atacar. Para solucionar esto, sería conveniente definir operaciones para modificar el conocimiento de agentes específicos:

- `in`([list-of-agents],t) que elimina toda posible derivación de la creencia t para los agentes en [list-of-agents].
- `rev`([list-of-agents],t) que revisa con respecto a t el conocimiento de los agentes en [list-of-agents]. Esta operación puede definirse mediante la secuencia de operaciones:

$$\texttt{in}([\text{list-of-agents}],\neg t);\ \texttt{out}(t)$$

7. Conclusiones y Trabajo Futuro

En este trabajo se definió una extensión al modelo Linda para representar un nuevo tipo de operaciones, `rev`, que permite agregar conocimiento de manera consistente, disparando procesos paralelos que restauran consistencia. La decisión de tomar el modelo Linda es su amplia aceptación en programación paralela, dado la simplicidad de su sintaxis y la ortogonalidad al lenguaje base en el que está embedido.

Los nuevos operadores propuestos son de gran utilidad ya que permitiría cambiar el comportamiento de los agentes sin reprogramar su función: simplemente, revisando el conocimiento referido a la estrategia de juego. Más aún, dada la generalidad de las operaciones propuestas, podría tanto modificarse la estrategia individual de cada agente como dar una estrategia global de equipo.

La operación `rev` permite definir la operación de revisión priorizada al estilo AGM, donde la nueva información siempre es aceptada. Como trabajo futuro, se planea aplicar este tipo de operadores para regular el comportamientos de los agentes del equipo Matebots (http://cs.uns.edu.ar/matebots) del Grupo de Robótica Cognitiva del Laboratorio de Investigación y Desarrollo en Inteligencia Artificial, que obtuvo el segundo puesto en la Robocup 2004.

Por último, se piensa definir un nuevo tipo de primitiva que permita representar operaciones no priorizadas donde, la nueva información puede ser aceptada, parcialmente aceptada o rechazada, dependiendo del grado de soporte o credibilidad de la misma. En ese sentido, los agentes tendrán la posibilidad de reaccionar de manera diferente ante el mismo cambio, en función de su propio conocimiento y de su forma de juego.

Referencias

[ACD90] V. Ambriola, P. Ciancarini, and M. Danelutto. Design and distributed inplementation of the parallel logic language Shared Prolog. *Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 40–49, 1990.

[AGM85] Carlos Alchourrón, Peter Gärdenfors, and David Makinson. *On the Logic of Theory Change: Partial Meet Contraction and Revision Functions*. *The Journal of Symbolic Logic*, 50:510–530, 1985.

[CG92a] Nicholas Carriero and David Gelernter. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[CG92b] Nicholas Carriero and David Gelernter. *How to write parallel programs: a first course*. MIT Press, 1992.

[FH94] André Fuhrmann and Sven Ove Hansson. *A Survey of Multiple Contractions*. *The Journal of Logic, Language and Information*, 3:39–76, 1994.

[FKIS02] Marcelo A. Falappa, Gabriele Kern-Isberner, and Guillermo R. Simari. *Belief Revision, Explanations and Defeasible Reasoning*. *Artificial Intelligence Journal*, 141:1–28, 2002.

[Fuh97] André Fuhrmann. *An Essay on Contraction*. Studies in Logic, Language and Information, CSLI Publications, Stanford, California, 1997.

[Gä88] Peter Gärdenfors. *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. The MIT Press, Bradford Books, Cambridge, Massachusetts, 1988.

[Han97a] Sven Ove Hansson. *Semi-Revision*. *Journal of Applied Non-Classical Logic*, 7:151–175, 1997.

[Han97b] Sven Ove Hansson. *Theoria: Special Issue on Non-Prioritized Belief Revision*. Department of Philosophy, Uppsala University, 1997.

[Han99] Sven Ove Hansson. *A Textbook of Belief Dymanics: Theory Change and Database Updating*. Kluwer Academic Publishers, 1999.

[HFCF01] Sven Ove Hansson, Eduardo Fermé, John Cantwell, and Marcelo Falappa. *Credibility Limited Revision*. *The Journal of Symbolic Logic*, 66(4):1581–1596, 2001.

[Lev77] Isaac Levi. *Subjunctives, Dispositions and Chances*. *Synthese*, 34:423–455, 1977.

[Mak97] David Makinson. *Screened Revision*. In **Theoria** [Han97b].

[Rob] Robocup official site. http://www.robocup.org/.

[Tar] Paul Tarau. Jinni 2002: A High Performance Java and .NET based Prolog for Object and Agent Oriented Internet Programming. http://www.cs.unt.edu/∼tarau/.

[Tar99] Paul Tarau. Intelligent Mobile Agent Programming at the Intersection of Java and Prolog. In *Proceedings of The Fourth International Conference on The Practical Application of Intelligent Agents and Multi-Agents*, pages 109–123, London, U.K., 1999.